

8/1/3

Docket No. AT9-99-149



PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of: **Carlson et al.**

Serial No. 09/329,456

Filed: June 10, 1999

For: **Method and Apparatus for
Monitoring and Handling Events for
a Collection of Related Threads in a
Data Processing System**

§
§
§
§
§
§
§

Group Art Unit: 2127

Examiner: **Tang, Kenneth**

RECEIVED

JUL 25 2003

Technology Center 2100

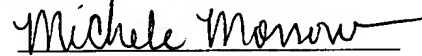
**Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450**

**ATTENTION: Board of Patent Appeals and
Interferences**

Certificate of Mailing Under 37 C.F.R. § 1.8(a)

I hereby certify this correspondence is being deposited with the United States Postal Service as First Class mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on July 18, 2003.

By:


Michele Morrow

APPELLANT'S BRIEF (37 C.F.R. 1.192)

This brief is in furtherance of the Notice of Appeal, filed in this case on March 4, 2003.

The fees required under § 1.17(c), and any required petition for extension of time for filing this brief and fees therefore, are dealt with in the accompanying TRANSMITTAL OF APPEAL BRIEF.

This brief is transmitted in triplicate. (37 C.F.R. 1.192(a)).

REAL PARTIES IN INTEREST

The real party in interest in this appeal is the following party:

International Business Machines Corporation of Armonk, New York.

RELATED APPEALS AND INTERFERENCES

With respect to other appeals or interferences that will directly affect, or be directly affected by, or have a bearing on the Board's decision in the pending appeal, there are no such appeals or interferences.

STATUS OF CLAIMS

A. TOTAL NUMBER OF CLAIMS IN APPLICATION

Claims in the application are: 1-28

B. STATUS OF ALL THE CLAIMS IN APPLICATION

1. Claims canceled: NONE
2. Claims withdrawn from consideration but not canceled: NONE
3. Claims pending: 1-28
4. Claims allowed: NONE
5. Claims rejected: 1-28

C. CLAIMS ON APPEAL

The claims on appeal are: 1-28

STATUS OF AMENDMENTS

All of the amendments to the claims have been entered. No after final amendments were made in this case.

SUMMARY OF INVENTION

The present invention relates to a system and method in data processing system for monitoring a plurality of related threads. (p. 11, ll. 12-15; p. 23, ll. 4-10; Figures 2-7). The plurality of threads is polled for status information. (p. 19, ll. 31 to p. 20, ll. 14; p. 24, ll. 20 to p. 26, ll. 3; Figures 6-7). Responsive to receiving the status information, a determination is made as to whether a thread within a plurality of related threads is active. (p. 23, ll. 4-30; Figure 7). Responsive to an absence of a determination that a thread within the plurality of related threads is active, a cleanup process is initiated for the thread based on the status information. (p. 23, ll. 22-30; p. 24, ll. 20 to p. 25, ll. 5; Figures 5 and 7).

ISSUES

The issues on appeal are:

1. Whether claims 1-3, 6, 11, 13-16, 19, 24, and 26-28 are obvious under 35 U.S.C. § 103(a) as being unpatentable over *Yeager* (US Patent No. 6,418,542).
2. Whether claims 9, 10, 12, 22, 23, and 25 are obvious under 35 U.S.C. § 103(a) as being unpatentable over *Yeager* (US Patent No. 6,418,542) in view of *Win* (US Patent No. 6,182,142).
3. Whether claims 5 and 18 are obvious under 35 U.S.C. § 103(a) as being unpatentable over *Yeager* (US Patent No. 6,418,542) in view of *Nation* (US Patent No. 6,233,599).
4. Whether claims 7 and 20 are obvious under 35 U.S.C. § 103(a) as being unpatentable over *Yeager* (US Patent No. 6,418,542) in view of *Anschuetz* (US Patent No. 5,305,455).
5. Whether claims 8 and 21 are obvious under 35 U.S.C. § 103(a) as being unpatentable over *Yeager* (US Patent No. 6,418,542) in view of *Yee* (US Patent No. 5,471,576).
6. Whether claims 4 and 17 are obvious under 35 U.S.C. § 103(a) as being unpatentable over *Yeager* (US Patent No. 6,418,542) in view of *Cejtin* (US Patent No. 5,745,703).

GROUPING OF CLAIMS

The claims stand or fall together as a single group.

ARGUMENT

I. Rejection of claims 1-28 under 35 U.S.C. § 103(a), Obviousness

Claims 1-3, 6, 11, 13-16, 19, 24, and 26-28 are rejected under 35 U.S.C. § 103(a) as being obvious in view of *Yeager* (US Patent No. 6,418,542). This rejection is respectfully traversed. Specifically, the Examiner states:

Referring to claim 1, 11, 14, 24, 27, and 28, Yeager discloses a data processing system for monitoring a plurality of related threads with the following steps:

polling the plurality or related threads for status information (“polling thread”) for “input events”, col. 2, lines 54-59 and “plurality of threads”, col. 3 lines 35-41);

determining whether a thread within a plurality of related threads is active as a response to receiving status information (“multi-threaded process” contains “information on each thread in the plurality of threads,” col. 3, lines 37-39, and “active”, “threads”, “critical signal” determines if active, col. 3, lines 16-22);

initiating cleanup processes for the thread based on the status information (“cleaning up a particular thread’s resources;”, “critical signal handler” decides when to clean based on its access to the “data in shared memory”, col. 8, lines 38-46).

Yeager fails to explicitly teach:

performing the cleanup process in response to an absence of determination that a thread within the plurality of related threads is active.

However, it would have been obvious to one of ordinary skill in the art at the time the invention was made to clean up errors whenever they occur, or more specifically, perform a “cleanup” when the system fails to determine if the threads are active. This would increase the efficiency because in the instance of being active, the unnecessary resource could be used by other threads.

(Office Action, dated August 27, 2002, pages 4-5).

For an invention to be prima facie obvious, the prior art must teach or suggest all claim limitations. *In re Royka*, 490 F.2d 981, 180 USPQ 580 (CCPA 1974). Independent claim 1 recites the features of polling a thread for status information, determining whether a thread is inactive in response to receiving the status information, and responsive to a determination that the thread is inactive, initiating cleanup processes for the thread based on the status information. Independent claim 1, which is representative of independent claims 11, 14, 24, 27, and 28, is reproduced below:

1. A method in a data processing system for monitoring a plurality of related threads, the method comprising the data processing system implemented steps of:
polling the plurality of related threads for status information;
responsive to receiving the status information, determining whether a thread within a plurality of related threads is inactive; and
responsive to a determination that a thread within the plurality of related threads is inactive, initiating cleanup processes for the thread based on the status information.

The claimed invention teaches polling a thread for status information. The Examiner states that *Yeager* teaches this feature in the following passages:

In yet another embodiment an input polling thread contained within the process is instructed to discontinue polling for input events directed to the offending thread. In yet another embodiment a core file of the process is made at the time it receives the critical signal even though the entire process is not shut down.

(*Yeager*, col. 2, ll. 54-59).

In yet another embodiment the system includes a memory shared by multi-threaded process in the system which contains information on each thread in the plurality of threads. In yet another embodiment the system includes an input polling thread in the process which is instructed by the critical signal thread to discontinue polling for input events directed to the offending thread.

(*Yeager*, col. 3, ll. 35-41).

These passages from *Yeager* do not teach or suggest polling a thread for status information. Instead, *Yeager* teaches handling an input event directed to a thread within a process operating in a multi-threaded system. The process is alerted that an input event affecting one of its active

connection threads has been received. A special thread referred to as an input polling thread is enabled and is used to determine which of the threads in the process has an event directed to it. That thread is triggered to handle the input event. However, *Yeager* fails to teach or suggest polling a thread for status information as in the claimed invention, because *Yeager* is directed toward polling for input events (events directed to a thread):

At step 314, an input polling thread is instructed to no longer poll on the offending thread. The input polling thread is described in greater detail in co-pending application Ser. No. 09/067,546, entitled "METHOD AND APPARATUS FOR DETECTING INPUT DIRECTED TO A THREAD IN A MULTI-THREADED PROCESS," which is incorporated herein by reference. In the described embodiment, a process has an input polling thread that detects and routes input events directed to the process to the appropriate thread in the process and eliminates the need for each thread in the process to be actively looking for input events directed to it.

(*Yeager*, col. 8, ll. 55-66). Even though *Yeager* teaches polling a thread, *Yeager* does not teach or suggest polling a thread *for status information* as in the claimed invention. Instead, *Yeager* teaches polling a thread *for input events*. "Input events," as used in *Yeager*, differs from "status information," as used in the present invention. Input events are generally some type of activity generated by a user or some external source, such as another computer network, which are directed to a thread. Polling of the input events involves detecting and routing those input events to the appropriate thread. "Status information," on the other hand, is merely the status of the thread.

In addition, an input event may itself affect the status information of a thread. An input event can cause the thread's status information to change in response to the event received. Thus, for the reasons set forth above, the polling process in *Yeager* fails to teach or suggest to one of ordinary skill in the art the step of polling a thread for status information as claimed in the present invention.

Furthermore, the claimed invention teaches determining whether a thread is inactive in response to receiving status information. The Examiner refers to the following passages as evidence that *Yeager* teaches the feature of determining whether a thread is active:

In another aspect of the present invention a computer system having a multi-threaded process capable of executing active connection threads where the system is arranged such that when a critical signal is generated for an offending thread, other

threads continue within the process is described.

(*Yeager*, col. 3, ll. 16-22).

In yet another embodiment the system includes a memory shared by multi-threaded process in the system which contains information on each thread in the plurality of threads. In yet another embodiment the system includes an input polling thread in the process which is instructed by the critical signal thread to discontinue polling for input events directed to the offending thread.

(*Yeager*, col. 3, ll. 34-41).

Nothing is present in these cited passages that teaches or suggests determining whether a thread is inactive in response to receiving status information. The passages above refer to allowing non-offending threads in a multi-threaded process to continue to function despite the presence of an offending thread. However, even assuming *arguendo* that the Examiner is correct in asserting that *Yeager* discloses that a critical signal determines the activity of a thread (“critical signal determines if active” from column 3, line 16-22), this interpretation still does not demonstrate that *Yeager* teaches or suggests the present invention. Even if *Yeager* determines if a thread is active or inactive, the reference does not teach the feature of determining whether a thread is inactive *in response to receiving status information*. Therefore, even assuming, for the purpose of argument, that *Yeager* teaches that a critical signal can determine if a thread is active or inactive, *Yeager* still fails to provide any teaching or suggestion that the determination that a thread is inactive is performed in response to receiving status information.

Moreover, the claimed invention in claim 1 initiates a cleanup process for a thread based on the status information and in response to a determination that a thread is not active. *Yeager* fails to teach or suggest this step. Although *Yeager* performs a cleanup process on a thread, *Yeager* does not teach or suggest performing the cleanup process in response to a determination that the thread is inactive. Figure 2 illustrates the *Yeager* cleanup process:

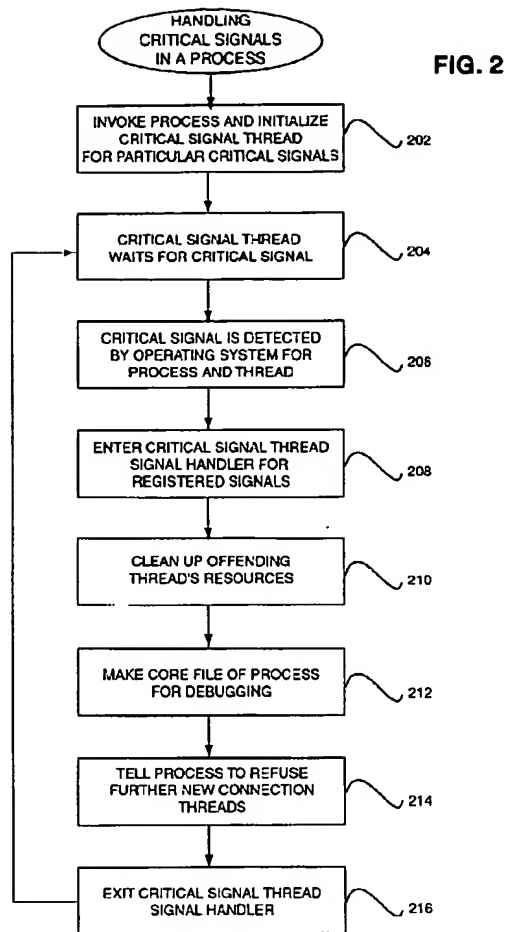


FIG. 2 is a flowchart showing a method of terminating an offending thread in a process without terminating the entire process in accordance with one embodiment of the present invention. At step 202 a process is executed and a critical signal thread associated with the process is initialized. When the process is started, it registers certain signals that it will catch and process. These signals include normal processing signals and critical signals that indicate an interrupt. For example, in the Solaris Operating System from Sun Microsystems, signals 10 and 11 are critical signals or interrupts indicating an illegal instruction and illegal memory access. When a signal is registered by a process at start time, the critical signal thread is enabled to handle that signal. Thus, at process start-up, the critical signal thread is enabled to catch any critical signals registered by the process. With previous systems, the process ignored critical signals and caught only normal or non-disruptive signals. When the signal is ignored, the operating system terminates the

process. In the described embodiment, the signals are registered by the process and the critical signal thread is notified to catch those signals.

(*Yeager*, col. 6, ll. 37-57).

The critical signal thread has several function calls it can make during its execution. In the described embodiment, one such function is known as the critical signal handler. The signal handler is a portion of code of the critical signal thread that handles the incoming registered signal. In other preferred embodiments, the critical signal thread can contain more than one critical signal handler for handling different types of critical signals. As mentioned above, in the Solaris Operating System signals 10 and 11 are critical signals that are handled by the same signal handler. In other preferred embodiments, there may be more than one signal handler for handler such signals. At step 208, the critical signal handler is invoked. When a signal is received by the crash thread from a signal queue maintained by the operating system, the thread calls the appropriate registered signal handler for handling the incoming critical signal.

(*Yeager*, col. 7, ll. 5-20).

Once the signal handler function has been called, the thread begins cleaning up the offending thread's resources. In the described embodiment, this is done by calling another function of the crash thread referred to as thread clean-up. This is done at step 210. The critical signal thread has access to shared memory and can determine which functions need to be called. The process of cleaning up an offending thread's resources is discussed in greater detail in FIG. 3.

(*Yeager*, col. 7, ll. 26-33). As can be seen from Figure 2, *Yeager's* cleanup process is performed in response to receiving a critical signal caused by an offending thread. A critical signal indicates that the thread is not operating properly, such as an illegal instruction or illegal memory access. There is no teaching or suggestion in *Yeager* to initiate cleanup of a thread's resources simply in response to a determination that the thread is inactive as in the claimed invention.

Similarly, *Yeager* fails to teach or suggest performing a cleanup process based on the status information as recited in the claimed invention. The Examiner cites the following as evidence that *Yeager* teaches this feature:

At step 308 the critical signal handler determines which function calls to make in order to clean up the thread's remaining resources. Since the critical signal thread has access to all the data in shared memory, it can determine which function calls are necessary for cleaning up a particular thread's resources. At step 310 the critical signal handler causes the system to remove all references to the thread from the shared allocated memory 116, as shown in FIG. 1.

(*Yeager*, col. 8, ll. 38-46). The cited passage demonstrates that *Yeager* uses a critical signal handler to make function calls to cleanup an offending thread's resources. However, the passage makes no mention of basing the cleanup process on the status information. Thus, *Yeager* teaches initiating a cleanup process based on the receipt of a critical signal caused by an offending thread, rather than initiating a cleanup process based on the status information as claimed in the present invention.

Furthermore, *Yeager* does not teach the problem or its source. The present invention recognizes the problem of determining whether a thread within a multi-threaded process is inactive, and if so, cleaning up that inactive thread's resources to enable a more efficient use of the system's resources. In contrast, *Yeager* is directed towards a system that allows threads in a multi-threaded process to continue executing when a single thread within the process receives a critical signal and crashes. Results of a thread crashing typically causes the entire operating system to shut down, or if the thread crashes on a network server, brings down the entire network. *Yeager* teaches handling critical signals directed to a thread in the process and not having the entire process, which has other threads running in it, terminate. One of ordinary skill in the art would therefore not be motivated to modify the reference in the manner required to form the solution disclosed in the claimed invention.

Furthermore, the Examiner may not merely state that the modification would have been obvious to one of ordinary skill in the art without pointing out in the prior art a suggestion of the desirability of the proposed modification. The mere fact that a prior art reference can be readily modified does not make the modification obvious unless the prior art suggested the desirability of the modification. *In re Laskowski*, 871 F.2d 115, 10 U.S.P.Q.2d 1397 (Fed. Cir. 1989) and also see *In re Fritch*, 972 F.2d 1260, 23 U.S.P.Q.2d 1780 (Fed. Cir. 1992) and *In re Mills*, 916 F.2d 680, 16 U.S.P.Q.2d 1430 (Fed. Cir. 1993). The claimed invention includes the feature of performing a cleanup process in response to a determination that a thread within the plurality of threads is inactive. In contrast, *Yeager* teaches performing a cleanup process on a particular thread's resources, but does not teach or suggest initiating a cleanup process for an inactive thread. In fact, the Examiner states that *Yeager* explicitly fails to teach "performing the cleanup process in response to an absence of a determination that a thread within the plurality of related threads is active." (*Office Action*, page 5). The rationale given for the modification is not based

on the prior art. Instead, the Examiner states his personal opinion using hindsight and “increased efficiency” as the rationale. The fact that a thread is inactive does not mean that one would perform a cleanup process on it. For example, the thread may be alive but suspended and inactive. One would not necessarily desire to perform a cleanup process on that inactive thread. Thus, since *Yeager* does not teach performing the cleanup process in response to a determination that a thread is inactive, nor does the Examiner point to any reference that suggests this feature be combined with the contents of the *Yeager* reference, Applicants respectfully submit that *Yeager* cannot be modified to produce the claimed invention.

Furthermore, *Yeager* does not teach, suggest, or give any incentive to make the needed changes to reach the presently claimed invention. *Yeager* actually teaches away from the presently claimed invention because this cited reference performs its cleanup process based on the receipt of a critical signal caused by an offending thread, rather than performing its cleanup process in response to a determination that the thread is inactive. Absent the Examiner pointing out some teaching or incentive to implement *Yeager* to perform a cleanup process on a thread in response to on a determination that a thread is inactive, one of ordinary skill in the art would not be led to modify *Yeager* to reach the present invention when the reference is examined as a whole.

Moreover, *Yeager* teaches away from the claimed invention since *Yeager* teaches initiating a cleanup process based on the receipt of a critical signal caused by an offending thread, rather than initiating a cleanup process based on the status information obtained. Thus, one of ordinary skill in the art would not be motivated from the reference to make the changes necessary to derive the present invention from the reference’s teachings.

In addition, the Examiner may not make modifications to the prior art using the claimed invention as a model for the modifications. *In re Fritch*, 972 F.2d 1260, 23 U.S.P.Q.2d 1780, 1783-84 (Fed. Cir. 1992). “The mere fact that the prior art may be modified in the manner suggested by the Examiner does not make the modification obvious unless the prior art has suggested the desirability of the combination.” *Id.* In other words, unless some teaching exists in the prior art for the suggested modification, merely asserting that such a modification would be obvious to one of ordinary skill in the art is improper and cannot be used to meet the burden of establishing a *prima facie* case of obviousness. Such reliance is an impermissible use of

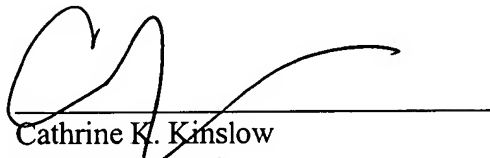
hindsight with the benefit of Applicants' disclosure. Absent some teaching, suggestion, or incentive in the prior art, *Yeager* cannot be properly modified to form the claimed invention; the presently claimed invention can be reached only through an impermissible use of hindsight with the benefit of Applicants' invention as a model.

If an independent claim is nonobvious under 35 U.S.C. §103, then any claim depending therefrom is nonobvious. *In re Fine*, 837 F.2d 1071, 5 USPQ2d 1596 (Fed. Cir. 1988). Claims 2-10, 12-13, 15-23, and 25-26 are dependent claims that depend on independent claims 1, 11, 14, 24, 27, and 28. Applicants have already demonstrated claims 1, 11, 14, 24, 27, and 28 to be in condition for allowance. Applicants respectfully submit that claims 2-10, 12-13, 15-23, and 25-26 are also allowable, at least by virtue of their dependency on allowable claims.

Therefore, the rejection of claims 1-28 under 35 U.S.C. § 103 has been overcome.

CONCLUSION

In view of the comments above, it is respectfully urged that the rejection of claims 1-28 not be sustained.



Cathrine K. Kinslow
Reg. No. 51,886
Carstens, Yee & Cahoon, LLP
PO Box 802334
Dallas, TX 75380
(972) 367-2001

APPENDIX OF CLAIMS

The text of the claims involved in the appeal are:

1. (previously amended) A method in a data processing system for monitoring a plurality of related threads, the method comprising the data processing system implemented steps of:

polling the plurality of related threads for status information;

responsive to receiving the status information, determining whether a thread within a plurality of related threads is inactive; and

responsive to a determination that a thread within the plurality of related threads is inactive, initiating cleanup processes for the thread based on the status information.
2. (original) The method of claim 1 further comprising:

responsive to receiving the status information, storing the status information.
3. (original) The method of claim 1, wherein the polling, determining, and initiating steps are performed by a single thread.
4. (previously amended) The method of claim 1, wherein the single thread is part of a class.
5. (original) The method of claim 1, wherein the initiating step comprises:

identifying active threads within the plurality of related threads;

identifying inactive threads within the plurality of related threads; and
terminating inactive threads.

6. (original) The method of claim 1, wherein the step of terminating inactive threads includes:

resetting resources allocated to an identified inactive thread such that the resources are reallocatable.

7. (original) The method of claim 1, wherein the plurality of related threads is a plurality of printer threads.

8. (original) The method of claim 1, wherein the plurality of related threads is a plurality of video threads.

9. (original) The method of claim 1, wherein the method is implemented in a virtual machine.

10. (original) The method of claim 9, wherein the virtual machine is a Java virtual machine.

11. (previously amended) A method in a data processing system for monitoring a plurality of related threads, the method comprising the data processing system implemented steps of:

polling the plurality of related threads for status information;

responsive to receiving the status information, determining whether a thread within a plurality of related threads is inactive; and

responsive to an occurrence of inactivity in a thread within the plurality of related threads in which the inactivity is due to an event, initiating cleanup processes based on the status information.

12. (previously amended) The method of claim 11, wherein the event is an occurrence of a period of time.

13. (original) The method of claim 11, wherein the event is an error.

14. (previously amended) A data processing system for monitoring a plurality of related threads, the data processing system comprising:

polling means for polling the plurality of related threads for status information;

determining means, responsive to receiving the status information, for determining whether a thread within a plurality of related threads is inactive; and

initiating means, responsive to a determination that a thread within the plurality of related threads is inactive, for initiating cleanup processes for the thread based on the status information.

15. (original) The data processing system of claim 14 further comprising:

storing means, responsive to receiving the status information, for storing the status information.

16. (original) The data processing system of claim 14, wherein the polling, determining, and initiating means are preformed by a single thread.

17. (previously amended) The data processing system of claim 14, wherein the single thread is part of a class.

18. (original) The data processing system of claim 14, wherein the initiating means comprises:

first identifying means for identifying active threads within the plurality of related threads;

second identifying means for identifying inactive threads within the plurality of related threads; and

terminating means for terminating inactive threads.

19. (original) The data processing system of claim 14, wherein the means of terminating inactive threads includes:

resetting means for resetting resources allocated to an identified inactive thread such that the resources are reallocatable.

20. (original) The data processing system of claim 14, wherein the plurality of related threads is a plurality of printer threads.

21. (original) The data processing system of claim 14, wherein the plurality of related threads is a plurality of video threads.

22. (original) The data processing system of claim 14, wherein the data processing system is implemented in a virtual machine.

23. (original) The data processing system of claim 22, wherein the virtual machine is a Java virtual machine.

24. (previously amended) A data processing system for monitoring a plurality of related threads, the data processing system comprising:

polling means for polling the plurality of related threads for status information;

determining means, responsive to receiving the status information, for determining whether a thread within a plurality of related threads is inactive; and

initiating means, responsive to an occurrence of inactivity in a thread within the plurality of related threads in which the inactivity is due to an event, for initiating cleanup processes based on the status information.

25. (previously amended) The data processing system of claim 24, wherein the event is an occurrence of a period of time.

26. (original) The data processing system of claim 24, wherein the event is an error.

27. (previously amended) A computer program product in a computer readable medium for monitoring a plurality of related threads, the computer program product comprising:

first instructions for polling the plurality of related threads for status information;

second instructions for responsive to receiving the status information, determining whether a thread within a plurality of related threads is inactive; and

third instructions for responsive to a determination that a thread within the plurality of related threads is inactive, initiating cleanup processes for the thread based on the status information.

28. (previously amended) A computer program product in a computer readable medium for monitoring a plurality of related threads, the computer program product comprising:

first instructions for polling the plurality of related threads for status information;

second instructions, responsive to receiving the status information, for determining whether a thread within a plurality of related threads is inactive; and

third instructions, responsive to an occurrence of inactivity in a thread within the plurality of related threads in which the inactivity is due to an event, for initiating cleanup processes based on the status information.